IPSec/PHIL (Packet Header Information List): Design, Implementation, and Evaluation

Chien-Lung Wu, NC State University, Raleigh, NC
S. Felix Wu, University of California, Davis, CA
Ravindar Narayan, Cosine Communication, Red Wood City, CA

Abstract

For most TCP/UDP/IP applications, when a packet or a message arrives, usually only the payload portion of the original packet can be obtained by the application. For instance, if a packet has been delivered through some IPSec tunnels along the route path, then the application, in general, will not know exactly which tunnels have been used to deliver this particular packet. The IPSec/PHIL (Packet Header Information List) interface has been designed and implemented such that an "authorized" application is able *to know* which set of IPSec tunnels has been used to deliver a particular incoming packet. Furthermore, IPSec/PHIL enables the controllability over which set of IPSec tunnels will be used to send a particular outgoing packet. IPSec/PHIL is a key component in the DECIDUOUS decentralized source tracing system to correlate the IPSec information with intrusion detection results. Other IPSec/PHIL applications we have built include a SNMPv3 security module using IPSec as well as a IPSec tunnel switching router.

1 Introduction

IP security (IPSec) protocol suite [1, 2, 3, and 4] is a series of guidelines for the protection of Internet Protocol (IP) communications. It provides ways for securing private information transmitted over public networks. The currently available IPSec-based applications in the market are predominantly Virtual Private Networks (VPNs). VPNs provide Network-to-Network security by setting up SAs (Security Associations) in the tunnel mode between Gateways of the networks, and these tunnels secure the aggregated data flowing from one policy domain to another through IPSec gateways.

For most TCP/UDP/IP applications, when a packet or a message arrives, usually only the payload portion of the original packet can be obtained by the application. If a packet has been delivered through some IPSec tunnels, then the application, in general, will not know exactly which tunnels have been used to deliver this particular packet. For instance, an intrusion source tracing system (such as DECIDUOUS [5, and 6]) might be very interested in analyzing not only the payload but also which particular IPSec tunnels have been used to deliver these attack packets with obviously spoofed source IP addresses.

For application-layer protocols such as SNMP and LDAP, it is usually not natural and feasible to use IPSec to secure the application-layer traffic, and thus a separate security mechanism is needed. In this paper, we will show that, with a simple extension of the socket API, the security mechanisms and capabilities of IPSec can support the security requirements of some applications. We called this new interface: IPSec/PHIL (Packet Header Information List [7]) API.

A third issue is regarding the support of end-to-end security using IPSec, while it is impossible to directly build a IPSec security association from the source to the destination. For instance, in an inter-domain environment, it might not be always possible to negotiate directly between two IP nodes belonging to two different domains. Things get trickier if an intermediate gateway will perform network address translation (NAT). We will show later that how to utilize the PHIL API to support "packet switching" among a set of IPSec tunnels such that it is possible to use a set of tunnels collaboratively (an IPSec tunnel path, more specifically) to secure the information end-to-end.

2 Background

IPSec protocol suite [8] has been discussing and developing in IETF IPSec working group. The fundamental concept of IPSec is to provide authenticated or security IPSec tunnel, such that any packet, going through this tunnel, has the confidentiality to verify that the packet is real be authenticated by the routers of the both end of the tunnel. If a packet goes through this tunnel, but not been authenticated will be dropped.

In IPSec, there are two types of protocols in doing tunnel authentication:

a. Authentication Header (AH) protocol provides support for data integrity and authentication of IP packets.
b. Encapsulating Security Payload protocol provides confidentiality services, including confidentiality of message contents, and also provides options for authentication of IP header.

Actually, IPSec protocol suite provides a flexible framework. For example, on AH protocol, users are allowed to use different AH algorithms—MD5, HMAC-MD5, ...,etc. And for ESP protocol, the ESP algorithm could be triple DES or IDEA.

For IPSec to do Authentication/Encryption, algorithms are important; but the key management is also very important as well. IPSec also provides a flexible way for users to do key

exchange. Currently IPSec adapts ISAKMP protocol to do security information exchange.

A key concept in both AH and ESP protocols is the Security Association (SA). An SA can be identify by three parameters:

1. Security Parameters Index (SPI): A SPI is a number assigned to a security association. The SPI is carried in AH and ESP headers to enable receiving system to select the SA when it received a packet.
2. IP Destination Address: the destination endpoint of A SA could be an end user or a network system such as a firewall or router.
3. Security protocol Identifier: This indicates whether the SA is an AH association or ESP association.

All SA are storage in Security Association database (SAdb). The SAdb has the relationship with Security policy. Security policy describes how this router will treat this packet. For example, the incoming packets could be dropped, could be routed without SA, or could be routed using another SA. All related security policies are stored in Security Policy database (SPdb).

Fig 1 is a simple example to see how IPSec work. First, from policy server, system administrators are able to update security policy between router A and B. The policy could be: " Any IP packet, going through A, will be encrypted (ESP protocol) and sent to B."

With this security policy, router A and B will look at their SAdb. If there existed the requested SA, then any packet, satisfying the security policy, will go through the authenticated tunnel using that SA. If there doesn't exist any available SA, both A and B will negotiate to decide which ESP algorithm is available for both A and B. After the negotiation, both A and B will agree to use SA (SPI=100) to encrypt all IP traffic.

In our research, instead of interested in the data integrity, we are more interested in the IP packet header authentication. For example, if we set up all policies between A and B, then if B received a packet, B has the confidence to say that the

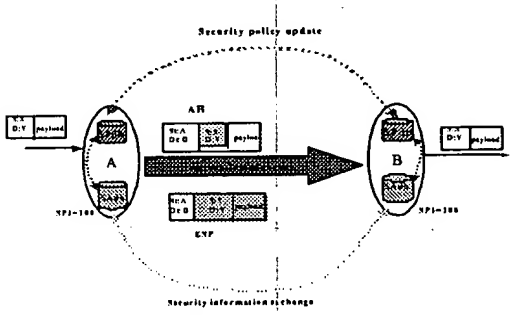

Fig 1: a simple example to show how IPSec work.

packet is actually from router A, even the Source address X is not trusted. Otherwise the packet will be dropped within B.

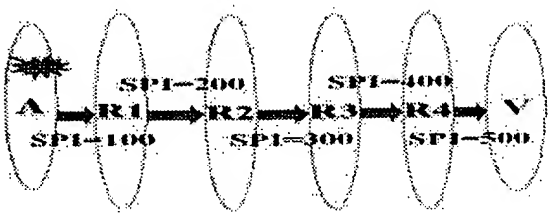

Fig 2: Applying IPSec to do attack source tracing

With the merit of authenticated tunnel, we come out the idea to do source identification and attack source tracing. For example, in Fig 2, if victim domain received a packet with the SPI=500, we can sure that the packet came from R4 (tracing back to previous router). Similarly, if R4 received a packet with SPI=400, he will know that the packet is from R3. Following the same process, we finally are able to know that the attacker is at A.

The thing interested is that "if a router receive a IP packet, can he know which SA has been used?" The answer is "No", he can just know that the packet has went through authenticated tunnel; but he didn't have any opportunity to know which SA has been used, and he even don't know this packet went through which authenticated tunnel, if there are multiple tunnel connected with this router. So how can we do? To solve this problem, we should have a way to keep all IPSec information (SA, SPI...) after the IPSec processing. This is the motivation for us to design PHIL (Packet Header Information List). Furthermore, if end-users are able to access or choice a SA to secure their communication, we can ext end the IPSec to support end-to-end communication as well.

3. Related works

Currently people using IPSec are more focus on the data confidentiality. Today there is still few research using the merit of IPSec authenticated tunnel to do attack source tracing yet. In 1993, Wobber and Lampson [9] proposed a theoretical concept of API for authentication. They suggested that the Operating System (OS) should provide an interface (API) for sending and receiving authenticated messages. D. McDonald [10] also proposed a draft of API extension to BSD socket. GRIP[11] (Gigabit Rate IP Security) project also has a study to extend API for support host-to-host connection. However, again, I got to emphasize that these proposals of API extension did not support security information keeping, since different purposes of using IPSec. As our study, to keep the security information is very important for us to use IPSec framework doing attack source identification. If we can keep the security information after IPSec process, it is nature for us to extend API to access the security information. Furthermore, if we have the design of API extension, it could

be possible for end-users to control IPSec security services in host-to-host connection.

4. Motivation

In the FreeSWAN implementation [12] of IPSec:

a. At the time of processing the **in-bound IPSec traffic**, all the IPSec headers are discarded at the IP layer; and when the applications do receive data, the data is devoid of all IPSec headers, thus there is no way of knowing whether the incoming packets were secured. If the packets were secured, then it is also hard to figure out what level of security was afforded and which end host or Security Gateways provided the security.

or UD

A TCP

b. Also, there is no way for user applications to control the **out-bound IPSec traffic** through a specific SA (security association). More specifically, since we are unable to bind an SA to a particular socket port in the application layer, we cannot support end-to-end application-layer security using IPSec.

Based on our observation and experience with IPSec, we believe that IPSec's capabilities can be greatly extended if we have a good interface to access the security services provided in the IP layer. Naturally, we would like to have the following two capabilities:

a. For incoming traffic it provides an API such that the application developers are able to extract security information such as the security afforded to a particular segment of data received at the application layer from the kernel.

b. For the outgoing traffic, PHIL-API provides the functionality to interact with the kernel's Security Association database (SAdB) and Security Policy database (SPdB) to query information about the existing SAs and the security level afforded to their outgoing data. It should also provide a way for the outgoing process to be able to override the default security policy.

5. Implementation

The key feature in PHIL is the "PHIL" information, which is a "list" data structure containing the IPSec related information. In regular protocol stack processing, all header information about IPSec has been stripped out before the payload being passed to the transport and application layers. However, in PHIL, extra IPSec information will be attached to the payload all the way up or down. The architecture of PHIL and its relationship with the OS kernel is depicted in the following figure. The detail implementation, please refer to [7].

5.1 PHIL-API for socket control

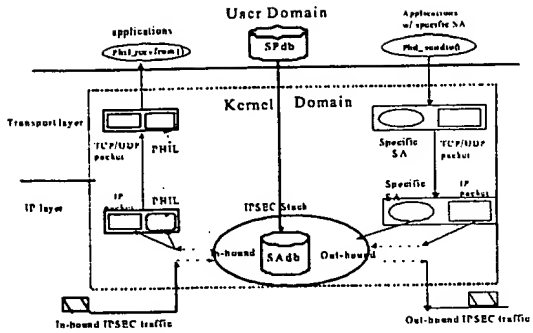

Figure 3: The Conceptual Architecture of PHIL-API

A TCP or UDP socket opened with a socket system call should first be enabled (or maybe later disabled) to receive the PHIL information along with the application data. The following functions are designed to control the PHIL functions:

int phil_enable *(int sockfd, int mode)*
int phil disable *(int sockfd)*
int phil bind *(int sockfd, unsigned long *spi array, int size)*
int phil unbind *(int sockfd)*
int phil_accept*(parameters to accept( ), char phil_buf, int phil_len) /* TCP only */*

5.2 For receiving data

int phil_recvfrom*(parameters to recvfrom( ), char *phil_buf, int phil_len, int *dsegs)*

In addition to the return values of the corresponding normal call ***recvfrom( )***, this call returns ***phil_buf***, which is a character buffer that contains the PHIL information, and ***dsegs***, which is the number of TCP data segments constituting the total data bytes being read from this call. A ***phil_recvfrom( )*** call is intended to be used to retrieve the data plus the PHIL information for both UDP and TCP applications. For sending data.

5.3 For sending data

int phil_sendto *(parameters to sendto( ), long *spi_arr , int size)*

The ***spi_arr*** array describes the SPI value(s) of our preference in sending the data (in the case where we have a choice of sending the data over several possible SAs). If the application does not know the set of the possible SA's, it can query the SAdB (Security Association database) for the SPI values through the query function of *spi( )* [7]. Through the ***phil_sendto( )***, it is possible to send a stream of data bytes from a single application process over different SAs, which provides different levels and features of security for different data types.

6. *Applications*

In today's TCP/UDP connection, it seems to people that there is not necessary of any PHIL support. However, in this section, we show that some applications really do need PHIL support, for example, Deciduous project of North Carolina State University use PHIL to resolve inter-domain tracing. And some applications, like SNMP, are able to use PHIL to simplify the security mechanism design. Following we will present several example to show how can we use PHIL to provide flexible controllability in offering security service.

6.1 PHIL Switching

Traditionally, an IP router will forward packets solely based on their destination addresses. In the DECIDUOUS project [5,6], in order to support "inter-domain collaboration," a router needs to switch the packets based on the "incoming" IPSec tunnels. Due to the space limitation, we will discuss very briefly how PHIL/PHIL switching technology is used in DECIDUOUS project. The functions of PHIL-switching can be summarized as:

1. Any incoming IPSec traffic, if matched any entry of the PHIL-switching table, will be forwarded to a specific security path.
2. If the incoming traffic is non-IPSec, it will be processed as normal IP traffic.
3. A selected set of inbound SA's can be aggregated into one outbound SA or dropped.

6.1.1 PHIL-Switching controller

The PHIL Switching controller provides an interface for users to add, delete or flush PHIL switching table. In our implementation, the controller has two different interfaces: one is a client-server model using UDP, the other is the SNMP MIB model. In the latter case, PHIL-switching is under the control of SNMP agent through the PHIL-switching MIB (Message Information Base). In either case, the user can "read, add, delete" and "flush" the PHIL-switching table.

6.1.2 User-Level Switching Entity

For any incoming packets with header fields such as [SPI, security protocol, source and destination addresses, protocol, source and destination ports], the switching entity looks up the PHIL switching table. Then, it will "switch" the incoming packets into different tunnels using specified SPI according to the switching table entry. To realize the concept of PHIL-Switching in the user level, we use "***divert socket***" to intercept IP packets from kernel to the user-level switching process. After the switching table look up, the intercepted packet will be forwarded using the ***phil_sendto( )*** with a specified SPI/SA

6.1.3. DECIDUOUS Collaboration protocol

In an intra-domain environment, it is easy to establish SA among routers. It is, in general, not possible to build up SA directly among any pair of routers in different domains. With the realization of PHIL-switching, DECIDUOUS can collaborate with a security gateway in another domain to establish an IPSec SA tunnel path, which emulates a direct SA across multiple domains.

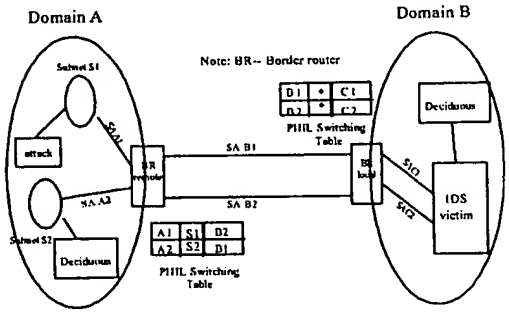

Figure 4: Inter-domain collaboration using PHIL-switching

We assume that SA can only be established between the border security gateways of collaborating domains. In Figure 4, six SA's (A1, A2, B1, B2, C1, C2) have been established between victim and local border router, between remote and local border routers, and within the remote domain. Now if IDS (Intrusion Detection System) detects attacks, it will report the detected attacks to local DECIDUOUS process. In the report, it will show that the attacks have been launched through SA C2. With the local PHIL-Switching table, we can tell that the attacks are indeed from SA B2. And, finally, the remote domain will be notified and it can further track down the source by correlating SA B2 with SA A2.

6.2 SNMP over IPSEC

The earlier versions of SNMP (SNMPv1 and SNMPv2) [13,14, and 15] use the **community** feature for a simple and unsecured password-based authentication. To improve the security concern, therefore, SNMPv3 introduces the concepts of ***snmpEngineId*** and ***securityName***. ***snmpEngineId*** uniquely identifies an SNMP engine that provides services for sending and receiving messages, authenticating and encrypting messages, and controlling access to managed objects. ***securityName*** is a human readable string representing an individual on whose behalf the services are provided or processed. Each ***securityName*** is associated (or configured) with a ***securityLevel*** parameter, which is stored in the ***Local Configuration Database (LCD)***. When a user issues a command or requests information, LCD is queried to

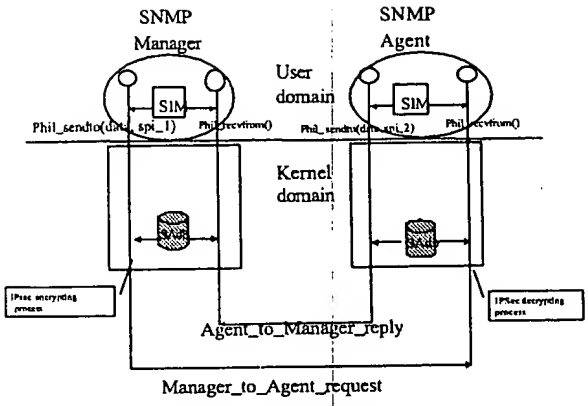

Figure 5: SNMP security architecture using IPSec

determine the security requirements for the given Figure 4: SNMP security architecture using IPSec ***securityName*** and ***snmpEngineID***. If the securityLevel specifies that the message is to be authenticated, then the message is authenticated according to the user's authentication protocol. Privacy and timelines modules are called depending on the ***securityLevel***.

For supporting SNMPv3 security on IPSec, first, we modified the SNMPv3 to support security information mapping (SIM) as shown on Figure 5. For example, The ***securityName***(Bill) and ***snmpEngineID***(Earth1) will generate two SPIs: SPI 0x161 for incoming, and SPI 0x171 for outgoing. Hence the SNMP packet data unit from SNMP manager to SNMP agent will be encrypted/decrypted using SPI 0x171; and when SNMP agent receives the request from SNMP manager, SNMP agent will use ***securityName***(Bill) to associate with the replying message and the replying security information (SPI 0x 161). With the PHIL-API (***phil_sendto( )***, and ***phil_recvfrom( )***), we can apply the specified SPI (generated by SIM) in the IPSec process.

7. *Performance Evaluation*

In this section, we evaluate the latency introduced by PHIL. The major overhead introduced by PHIL is the PHIL process for both incoming side and outgoing side.

Environments and methods

The measurement includes two machines: both are 450MHz Pentium II equipped with 10Mbit/sec Ethernet cards, and run on Linux2.0.36 with FreeSwan 1.0[12]. We create echo client/server (TCP/UDP) to launch 100000 packets each times and then calculate the average time in each sending and receiving process. We also concern the different data sizes in 16, 32, 64, 128, 256, 512, 1024 bytes. For any packet great than MTU (Maximum Transmission Unit), this packet will be fragmented and the PHIL process could be executed more than twice (depends upon the data size) in each packet. Hence we restrict the packet size to be less than MTU such that we can simply calculate the average time delay for each packet. Three cases are under testing:

Case 1: Linux 2.0.36 kernel, no IPSec, and no PHIL.
Case 2: Linux 2.0.36 kernel with FreeSwan IPSec (V1.0); no PHIL.
Case 3: Linux 2.0.36 kernel with FreeSwan IPSec (V1.0) and PHIL.

The result is shown on table 7-1. From case1 and case 2, we show the IPSec overhead; and by comparing case2 and case3, we present the PHIL implementation overhead. Within Table 7.1, we observed that the overhead introduced by PHIL (compare with Case2 and Case 3) is 1-2 microseconds.

Table 7.1: The test result of PHIL

size(bytes)	Case 1	Case 2	Case 3	Case3-Case2
16	363 us	ESP: 505 us	ESP: 506 us	ESP: 1 us
		AH: 502 us	AH: 502 us	AH: 0 us
32	398 us	ESP: 538 us	ESP: 539 us	ESP: 1 us
		AH: 535 us	AH: 536 us	AH: 1 us
64	465 us	ESP: 616 us	ESP: 617 us	ESP: 1 us
		AH: 613 us	AH: 613 us	AH: 0 us
128	601 us	ESP: 761 us	ESP: 762 us	ESP: 1 us
		AH: 756 us	AH: 757 us	AH: 1 us
256	870 us	ESP: 1050 us	ESP: 1051 us	ESP: 1 us
		AH: 1045 us	AH: 1046 us	AH: 1 us
512	1410 us	ESP: 1628 us	ESP: 1630 us	ESP: 2 us
		AH: 1625 us	AH: 1625 us	AH: 0 us
1024	1410 us	ESP: 1628 us	ESP: 1630 us	ESP: 2 us
		AH: 1625 us	AH: 1625 us	AH: 0 us

8. *Remarks and Conclusions*

IPSec has been standardized and widely deployed for securing private information over the Internet. Currently, VPN is the major application for IPSec since higher layers cannot easily access and control IPSec-layer security services. Our PHIL-API design and implementation provides a possible bridge between IPSec and other security applications. We have demonstrated the usefulness of this new API for applications such as DECIDUOUS and SNMPv3. We believe that many other secure Internet applications can be built directly on top of IPSec/PHIL, without having to re-develop yet another security module within the application layer. Furthermore, in the near future, we expect to see more and more hardware acceleration for IPSec (e.g., 3Com's IPSec NIC, and CellTech's Gigabit IPSec chip). Therefore, for high performance applications, it will be much more attractive to use IPSec/PHIL than the software-based lower-throughput transport/application layer protocols such as TLS or SSL. Practically, the PHIL-API is useful when the applications need to run on platforms not supporting other security protocols (e.g., TLS). Finally, through our implementation and evaluation, we have shown that the overhead (memory space and CPU time) in providing PHIL is quite reasonable – 1 to 2 microseconds per packet in software. The PHIL service has been integrated into the DECIDUOUS system, which can trace the true attack sources in a few seconds on top of a 10-node test-bed.

9. *References*

[1] S. Kent, and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[2] S. Kent, and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 2406, November 1998.

[3] S. Kent, and R. Atkinson, "IP Authentication Header," RFC 2402, November 1998.

[4] Angelos D. Keromytis, John Ioannidis, and Jonathan M. Smith, "Implementing IPSec," IEEE, August 1997.

[5] H.Y. Chang, S.F. Wu, et al., "Deciduous: Decentralized Source Identification for Network-based Intrusions", appeared in 6th IFIP/IEEE International Symposium on Integrated Network Management, IEEE Communications Society Press, May 1999.

[6] H.Y. Chang, S.F. Wu, et al., "Design and Implementation of a Real-Time Decentralized Source Identification System for Un-trusted IP Packets", appeared in DARPA Information Survivability Conference and Exposition (DISCEX 2000), IEEE Computer Society Press, January, 2000.

[7] Ravindra Narayan, "Socket API Extensions to Extract Packet Header Information List (PHIL)" Master thesis, May 1999. http://www.lib.ncsu.edu/etd/public/etd-3721141949931381/etd-title.html

[8] IP Security working group of IETF: IP security roadmap http://www.ietf.org/rfc/rfc2411.txt

[9] Edward P. Wobber, Martin Abadi, Michael Burrows, and Butler Lampson. "Authentication in the Taos Operating System" *ACM Transactions on Computer Systems*, 12(1):3-32, February 1994. Also appeared as SRC Research Report 117

[10] D. McDonald, "draft-mcdonald-simple-ipsec-api-03.txt", http://www.alternic.org/drafts/drafts-m-n/draft-mcdonald-simple-ipsec-api-03.html

[11] GRIP (Gigabit Rtae IP Security): http://www.east.isi.edu/DIV10/GRIP/

[12] Linux IPSec/FreeSwan Web site: http://www.xs4all.nl/~freeswan/

[13] Harrington, D., Presuhn R., Wijnen B., "An Architecture for describing SNMP management frameworks", RFC 2271, January 1998.

[14] D. Levi, P. Meyer, B. Stewart, "SNMPv3 Applications," RFC2273, January 1998

[15] Blumenthal, U., Wijnen B., "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol", RFC 2274, January 1998.

[16] The Linux Kernel Archives: http://www.kernel.org/

[17] M. Beck, H. Bohme, M. Dziadzka, and U. Kunitz, "Linux Kernel Internals" Addison-Wesley Second version, 1998.